

Towards practical classical processing for the surface code: timing analysis

Austin G. Fowler, Adam C. Whiteside, Lloyd C. L. Hollenberg
 Centre for Quantum Computation and Communication Technology,
 School of Physics, The University of Melbourne, Victoria 3010, Australia
 (Dated: February 28, 2012)

We perform a timing analysis of our current implementation of the algorithm described in [1]. Our implementation performs the classical processing associated with an $n \times n$ lattice of qubits realizing a square surface code storing a single logical qubit of information in a fault-tolerant manner. We empirically demonstrate that our implementation requires only $O(n^2)$ average time per round of error correction for code distances ranging from 4 to 512 and a range of depolarizing error rates. We also describe tests we have performed to verify that it always obtains a true minimum weight perfect matching.

We assume familiarity with the surface code [2, 3], fault-tolerant schemes built on the surface code [4–6] and the classical processing algorithm described in [1]. Our goal is not to repeat existing discussion, rather to provide additional data on the performance and correctness of our implementation of this algorithm. This implementation is called *scode* (*ess-code*).

Scode consists of four layers of software — simulator, problem preparer, problem solver, data gatherer. The simulator executes no initialization surface code error detection with depolarizing noise (see [7]). Random Pauli errors are generated and propagated using a Pauli frame. When errors lead to syndrome measurement value changes, graph vertices are generated at these space-time locations by the problem preparer. By pre-analyzing all possible single error processes [7], an underlying lattice of dots and lines is also prepared with dots at every location a vertex could potentially be generated and lines between every pair of locations that could have vertices generated by a single error. The first order probability p_{line} of each line is calculated and a weight $w = -\ln(p_{\text{line}})$ stored in each line. The rationale for doing this can be found in [7]. A lattice of dots and lines with associated probabilistically generated vertices (from surface code simulation) is shown in Fig. 1.

In many ways, a lattice plus vertices can be considered an implicit complete graph with an edge between any pair of vertices having weight equal to the minimum weight path between those vertices. The task is to match all vertices in pairs or to neighboring boundaries such that the total weight of all match paths is minimal. The basic algorithm that efficiently solves this problem given a standard graph is the minimum weight perfect matching algorithm [8, 9]. We have extended this algorithm to include the concept of boundaries and permit new vertices to be dynamically added to the graph.

We have two operational versions of extended minimum weight perfect matching — complete match [7] which firstly constructs explicit edges between all pairs of vertices no more than approximately d rounds of error correction apart, and edges on demand match [1] which only constructs a small number of local edges and adds further edges to the problem as required. The graphs and matchings generated by complete match (*cmatch*)

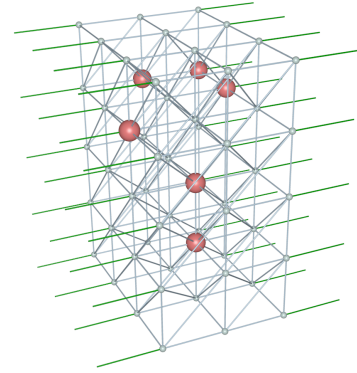


FIG. 1: Distance 4 example of a lattice of dots and lines with associated vertices. Dots (small silver balls) correspond to space-time locations where the endpoints of error chains could potentially be detected. Vertices (large red balls) correspond to space-time locations where error chain end points have been detected. Silver lines link pairs of dots where a pair of vertices could be generated by a single error. Green lines link spatial boundaries to a single dot where a single vertex could be generated by a single error.

and edges on demand match (*eodmatch*) given Fig. 1 as input are shown in Fig. 2 and Fig. 3 respectively. The total weight of matched edges in both cases is identical and in this case the matchings themselves are identical. We have tested *cmatch* and *eodmatch* on millions of varied problems, large and small, and always observed identical total weights, strongly implying both implementations are correct.

Cmatch obtains the true minimum weight perfect matching despite only including edges between vertices separated by a finite number of rounds. Vertices separated by a very large number of rounds are always cheaper to match to their nearest boundaries than to one another. By using the weights of the lines in the lattice, we calculate the minimum span of rounds to connect with edges to guarantee a minimum weight matching. *Eodmatch* also obtains a true minimum weight perfect matching as any required edge will eventually be included

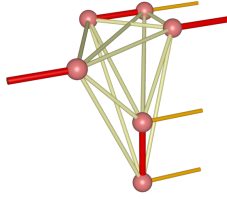


FIG. 2: Output of `cmatch` when given Fig. 1 as input. The underlying lattice is used to construct a complete graph and then discarded. Edges in the matching are shown in red.

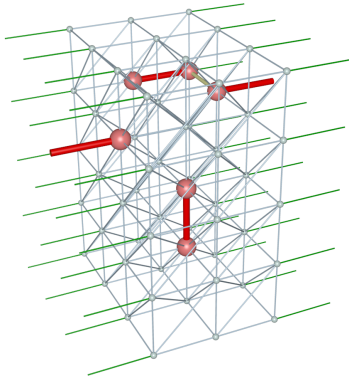


FIG. 3: Output of `eodmatch` when given Fig. 1 as input. Note that only one edge other than those ultimately included in the matching has been created (just visible between the top right two vertices).

during execution.

Further evidence of the correctness of `eodmatch` comes from studying the probability of logical error per round of error correction (p_L) at depolarizing probabilities p well below threshold. We calculate p_L by simulating `t.check` rounds of faulty quantum computer operation, then turning off errors, capping the matching problem with a perfect round of error correction, applying corrections, checking whether we have an odd or even number of errors along one of the boundaries and recording whether this is different to the previous time we checked. The perfect round of error correction is then undone and another `t.check` faulty rounds simulated and the process repeated.

It may seem that the ideal value of `t.check` is 1 to ensure that no logical errors are missed, however this is not the case. We have observed that many combinations of errors lead to the observation of a logical error if a perfect round of error correction is inserted halfway through it, but no logical error if the perfect round of correction is sufficiently distant. With frequent checking this can mean a benign pattern of errors is counted as several logical errors. Instead, we typically use a value of `t.check`

such that a change in the parity of the number of errors observed along a boundary occurs approximately 10% of the time. We have empirically found that this leads to a logical error rate estimate robust to wide variations of `t.check` about this value. The probability of a change per check is equal to the probability of an odd number of logical errors in `t.check` rounds enabling p_L to be easily calculated.

A distance d code can reliably correct $\lfloor (d-1)/2 \rfloor$ errors. At low error rates p clusters of errors are well separated. The probability of suffering a logical error inducing cluster of $n_d = \lfloor (d+1)/2 \rfloor$ errors should therefore be $O(p^{n_d})$ if the full distance of the code is being realized. Figs. 4–5 show the complete set of data we have collected for the square surface code. Polynomials $A_d p^{n_d}$ are drawn through the lowest data point we were able to obtain for distances 3, 5, 7 and 9.

It is computationally expensive to obtain statistics at very low error rates and high distances as very few logical state changes are observed. It is also computationally expensive to obtain data at high error rates and high distances as the minimum weight perfect matching problem becomes more difficult around and above the threshold error rate (0.9% [1]). The raw data used to generate Figs. 4–5, including timing information, can be found in Appendix A.

The distance 3 and 5 dashed asymptotic curves in Figs. 4–5 agree very well with the data. For higher distances, it is not currently possible to simulate a sufficiently large number of rounds of error correction to obtain sufficient information at low enough distances and achieve such tight agreement. Note that the high distance data curves approach the asymptotic curves with a steeper gradient, implying the surface code is capable of regularly correcting temporal clusters of errors containing more errors than the maximum guaranteed to be correctable. This is a generic feature of topological quantum error correction, as a large cluster of errors widely scattered across the code is not dangerous provided the errors are sufficiently sparse.

The timing information in Appendix A includes everything — initial bootup of the simulation, the simulation of the underlying quantum computer, problem generation, matching, perfect rounds of error correction to enable logical state change detection, and maintenance of an appropriate Pauli frame. Figs. 6 shows the amount of time devoted to each round of matching alone at three different error rates for distances $d = 4, 8, 16, \dots, 512$. The quadratic scaling of required time with distance is well demonstrated. At small d nearby boundaries prevent the growth of large blossoms leading to increased performance. At very high d memory access effects lead to a slight slowdown. Note that real computer systems are too complex to provide perfectly smooth graphs of time scaling even with long time averaging as the interplay of different levels of cache and RAM leads to measurable deviations from the ideal scaling.

To illustrate the complexity of modern computer mem-

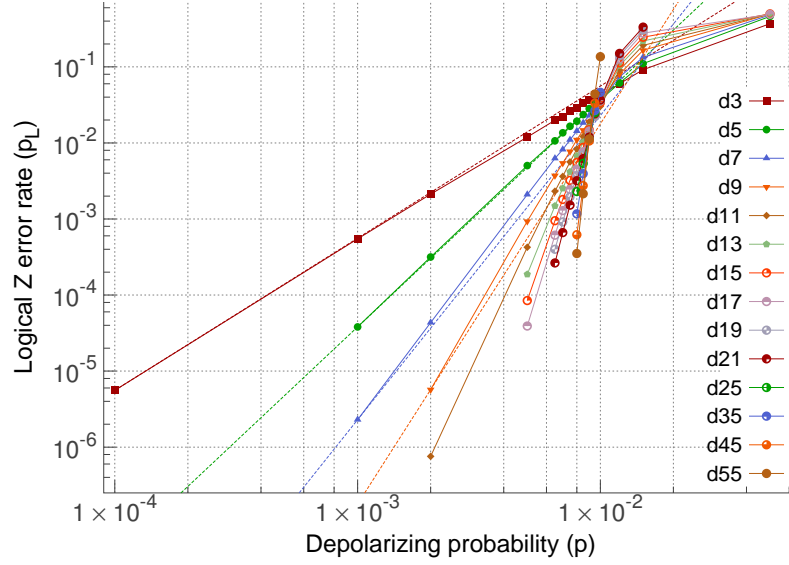


FIG. 4: Logical Z error rate per round of error correction for surface code distances d and depolarizing noise probabilities p . Dashed lines indicate expected low p asymptotic curves for $d = 3, 5, 7$ and 9 .

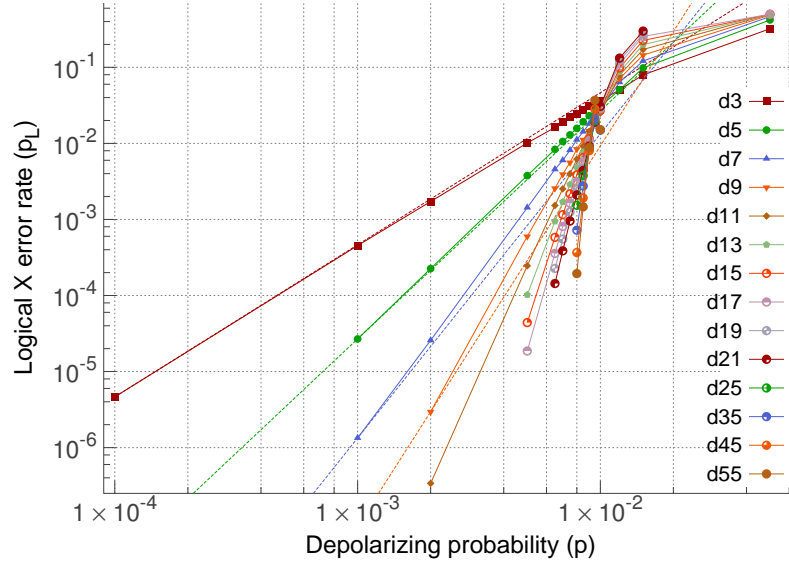


FIG. 5: Logical X error rate per round of error correction for surface code distances d and depolarizing noise probabilities p . Dashed lines indicate expected low p asymptotic curves for $d = 3, 5, 7$ and 9 .

ory systems, we have generated increasingly large arrays of random integers and calculated the time required to swap a constant large number (10^{11}) of randomly chosen pairs of integers. The results are shown in Fig. 7. Ideally, a swap operation should be $O(1)$ independent of the array size. In practice, it can be seen that larger data sets lead to lower performance as CPU cache is exceeded. The data in this manuscript was generated by 16 core Intel Xeon 3.33GHz CPUs with 12MB of cache. Our matching code is more complex than this simple demonstration, with gradual delocalization of data as the data

set increases in size. This leads to a gradual reduction of the probability of a single memory page load containing additional useful data.

To summarize, after accounting for low distance nearby boundaries which limit the complexity of matching and high distance slower memory access, Figs. 6 provide strong evidence supporting the claimed $O(d^2)$ runtime of our implementation of the algorithm described in [1].

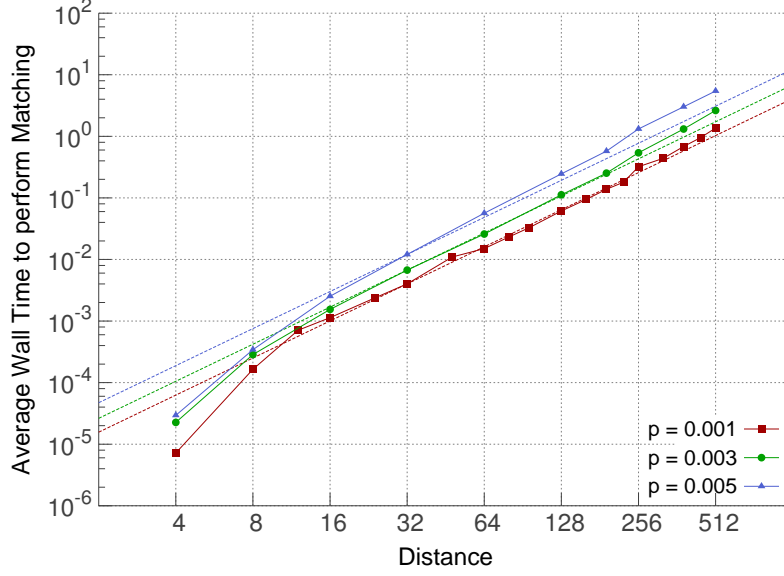


FIG. 6: Amount time in seconds devoted to each round of matching when simulating a distance d single logical qubit square surface code for depolarizing error rates p . Quadratic curves have been included for reference.

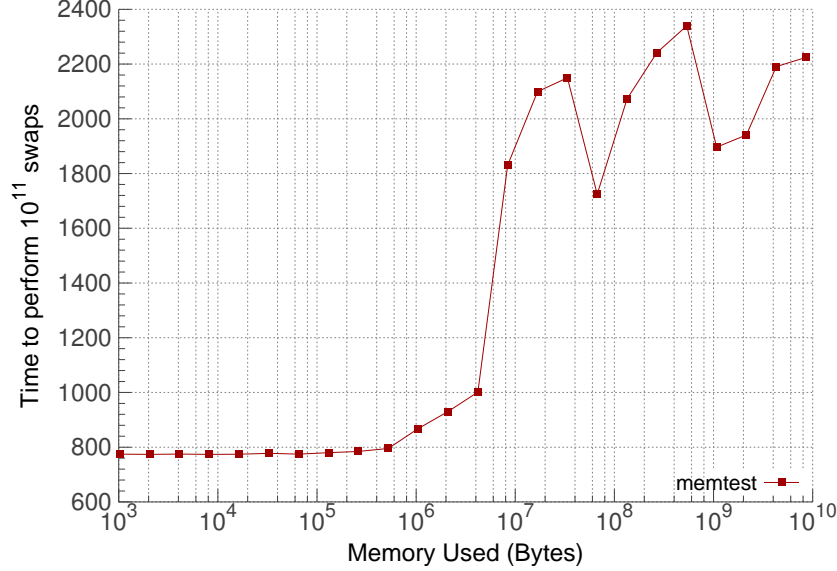


FIG. 7: Average time in seconds required to perform 10^{11} swaps of randomly chosen pairs of integers in arrays of increasing size.

I. ACKNOWLEDGEMENTS

We acknowledge support from the Australian Research Council Centre of Excellence for Quantum Computation and Communication Technology (Project number CE110001027), and the US National Security Agency (NSA) and the Army Research Office (ARO) under contract number W911NF-08-1-0527.

Appendix A: Raw data

The raw data used to generate Figs. 4–5, including timing information, is listed below. The first number is the number of different distances (14). The second is the distance d of the following block of data, the third is the number of different values n of the depolarizing error rate p . The next n lines list the value of p , t_{check} , the

number of checks for Z_L state changes, the number of checks for X_L state changes, the observed number of Z_L state changes, the observed number of X_L state changes and finally the total number of CPU seconds devoted to the simulation of that (d, p) pair. This basic structure is repeated for each distance. The last number in the file is the total number of CPU seconds devoted to creation of the entire file.

```

14
3
15
0.0001 25780 93997 93997 11754 10000 37685.82
0.001 261 94887 94886 11810 10000 415.42
0.002 70 93413 93413 12015 10000 228.69
0.005 12 91697 91696 11540 10000 33.62
0.0065 7 97198 97197 12015 10000 30.32
0.007 5 113854 113853 11598 10000 25.31
0.0075 4 121578 121577 11670 10000 24.84
0.008 3 142693 142692 11681 10000 26.08
0.0085 2 184890 184889 11830 10000 29.62
0.009 2 164349 164348 11696 10000 26.93
0.0095 3 107513 107513 11496 10000 28.59
0.01 3 98958 98957 11523 10000 21.47
0.012 2 103971 103970 11759 10000 22.25
0.015 1 125634 125633 11536 10000 27.58
0.05 1 31051 31050 11478 10000 28.47
5
14
0.001 3861 106980 106979 13628 10000 47490.43
0.002 314 151251 151251 13637 10000 6568.45
0.005 26 111948 111947 12953 10000 653.26
0.0065 9 142249 142248 12523 10000 416.50
0.007 6 165703 165702 12626 10000 359.61
0.0075 6 137318 137317 12553 10000 308.21
0.008 6 114309 114308 12021 10000 325.45
0.0085 4 137468 137467 12061 10000 329.63
0.009 3 151197 151196 12121 10000 336.85
0.0095 3 131364 131364 11914 10000 304.38
0.01 3 116216 116215 11896 10000 337.23
0.012 2 103659 103658 11950 10000 332.92
0.015 1 101838 101837 11265 10000 269.22
0.05 1 23807 23806 10975 10000 544.14
7
14
0.001 65493 124652 124652 16178 10000 3521570.27
0.002 3731 114602 114602 15837 10001 165464.09
0.005 88 89778 89777 13864 10000 6818.24
0.0065 13 178531 178530 13492 10000 3272.27
0.007 18 102587 102586 13184 10000 1624.85
0.0075 10 131935 131934 13115 10000 1651.60
0.008 6 157820 157819 12635 10000 1387.70
0.0085 5 148477 148476 12683 10000 1220.78
0.009 4 146237 146236 12497 10000 1470.92
0.0095 4 120724 120724 12117 10000 2172.96
0.01 3 126889 126888 12183 10000 1086.01
0.012 1 157182 157181 11622 10000 1510.58
0.015 1 82346 82345 11007 10000 1145.62
0.05 1 21522 21521 10388 10000 1717.54
9
13
0.002 31862 116333 116333 17719 10001 4289126.50
0.005 134 134774 134773 14912 10000 34427.08
0.0065 31 135372 135371 14001 10000 9139.10
0.007 17 159583 159582 13532 10000 6835.25
0.0075 13 146483 146482 13431 10000 5371.73
0.008 6 206359 206358 12936 10000 4678.65
0.0085 6 157382 157381 12947 10000 3371.82
0.009 6 122371 122370 12735 10000 2893.09

```

```

0.0095 3 163207 163207 12108 10000 2943.78
0.01 2 177919 177918 12051 10000 5089.92
0.012 1 140234 140233 11265 10000 4300.04
0.015 1 68981 68980 11432 10000 3114.11
0.05 1 20548 20547 10181 10000 5762.83
11
13
0.002 10000 2970715 2970715 22288 10014 60628087.09
0.005 442 102391 102390 16041 10000 116498.16
0.0065 39 177762 177761 14738 10000 27844.55
0.007 34 126474 126473 13882 10000 26712.36
0.0075 18 149239 149238 13749 10000 15172.28
0.008 10 170139 170138 13115 10000 9234.01
0.0085 8 144574 144573 12947 10000 10480.43
0.009 3 237671 237670 12947 10000 9925.01
0.0095 3 171695 171695 12202 10000 9361.08
0.01 4 107858 107857 11928 10000 6497.89
0.012 1 126752 126751 11515 10000 7390.47
0.015 1 58162 58161 11174 10000 4262.45
0.05 1 19984 19983 9936 10000 12520.50
13
12
0.005 937 114572 114572 16980 10000 603790.28
0.0065 58 192423 192422 15266 10000 78381.30
0.007 32 193240 193239 14502 10000 50053.45
0.0075 26 143566 143565 14055 10000 34005.52
0.008 12 180949 180948 13613 10000 22861.61
0.0085 10 139111 139110 13025 10000 21325.58
0.009 5 172414 172413 12920 10000 20014.77
0.0095 5 121057 121057 12400 10000 13410.58
0.01 3 130958 130957 12065 10000 11380.03
0.012 1 115078 115077 11931 10000 10931.53
0.015 1 50282 50281 11036 10000 6670.03
0.05 1 19964 19963 10009 10000 28043.39
15
12
0.005 2456 102892 102890 17492 10001 2173892.01
0.0065 151 124143 124143 15471 10000 222593.70
0.007 52 176282 176282 15132 10000 115430.02
0.0075 28 174215 174215 14415 10000 78240.36
0.008 17 161912 161912 14015 10000 52890.79
0.0085 11 148576 148576 13102 10000 54511.94
0.009 5 181422 181422 12771 10000 34858.01
0.0095 4 146241 146241 12416 10000 30968.27
0.01 3 129322 129322 12253 10000 28182.51
0.012 1 101864 101864 11498 10000 17875.60
0.015 1 44247 44247 10863 10000 11634.90
0.05 1 19864 19864 9903 10000 98384.02
17
12
0.005 5885 101613 101613 18732 10000 6704496.11
0.0065 275 111939 111939 16021 10000 516700.36
0.007 89 149029 149029 15277 10000 256345.82
0.0075 54 123138 123138 14291 10000 126590.64
0.008 23 148027 148027 13961 10000 82716.06
0.0085 15 127265 127265 13161 10000 56207.63
0.009 5 191569 191569 12961 10000 45131.54
0.0095 4 148226 148226 12348 10000 34119.88
0.01 2 169382 169382 12010 10000 46805.81
0.012 1 89938 89938 11408 10000 24677.61
0.015 1 39250 39250 10892 10000 17671.96
0.05 1 20013 20013 9818 10000 132420.21
19
10
0.0065 420 115409 115409 16474 10000 963439.12
0.007 163 121377 121377 15659 10000 533590.20
0.0075 69 125702 125702 14879 10000 269356.52
0.008 46 94467 94467 13698 10000 144521.98
0.0085 20 114779 114779 13936 10000 82840.47
0.009 8 142942 142942 12913 10000 60109.55

```

```

0.0095 4 146009 146009 12300 10000 55487.99
0.01 3 123422 123422 12369 10000 47540.48
0.012 1 81072 81072 11411 10000 35343.85
0.015 1 36705 36705 11159 10000 28088.66
21
10
0.0065 859 91469 91463 16689 10000 2147913.54
0.007 226 124800 124798 16152 10000 1191853.64
0.0075 107 108172 108172 15048 10000 472320.02
0.008 44 118209 118209 14480 10000 276634.69
0.0085 19 128841 128841 13654 10000 125697.36
0.009 8 144541 144541 12716 10000 93677.72
0.0095 3 182294 182294 12630 10000 170367.06
0.01 3 115647 115647 11821 10000 63634.76
0.012 1 75689 75689 11377 10000 52612.18
0.015 1 33355 33355 11109 10000 40737.04
25
5
0.008 55 128564 128564 14352 10000 507860.33
0.0085 17 165192 165192 13888 10000 231570.51
0.009 9 143332 143332 13211 10000 175110.91
0.0095 3 173140 173140 12352 10001 163829.39
0.01 1 9 9 0 1 1.04
35
5
0.008 88 167587 167587 15592 10000 2801803.43
0.0085 14 269115 269115 14131 10000 1292528.08
0.009 7 176942 176942 12822 10000 550365.22
0.0095 4 125106 125106 12314 10001 437366.43
0.01 2 34 34 3 1 61.08
45
5
0.008 230 128455 128422 15969 10000 11334092.96
0.0085 27 203196 203196 14039 10000 2845252.53
0.009 9 147701 147701 12962 10000 1587903.33
0.0095 3 125196 125196 11684 10001 944033.23
0.01 1 13 13 0 1 16.30
55
5
0.008 197 285003 284944 18433 10489 37296797.55
0.0085 34 210799 210785 14340 10000 7669758.24
0.009 7 170072 170072 12541 10000 3025099.15
0.0095 2 140843 140840 11836 10000 2813159.55
0.01 1 66 66 9 1 2038.71
161469013.53

```

For convenience, we also include below the processed raw data with checks and changes converted into probabilities of logical error. Second column is the probability of Z_L error per round of error correction. Third column is the same data for X_L .

```

14
3
15
1.000000e-04 5.581956e-06 4.639970e-06
1.000000e-03 5.480776e-04 4.532594e-04
2.000000e-03 2.119693e-03 1.717972e-03
5.000000e-03 1.193646e-02 1.014745e-02
6.500000e-03 1.987923e-02 1.618780e-02
7.000000e-03 2.227094e-02 1.894943e-02
7.500000e-03 2.594789e-02 2.196915e-02
8.000000e-03 2.892852e-02 2.454551e-02
8.500000e-03 3.308666e-02 2.781694e-02
9.000000e-03 3.694796e-02 3.140969e-02
9.500000e-03 3.853583e-02 3.315384e-02
1.000000e-02 4.229061e-02 3.624911e-02
1.200000e-02 6.016994e-02 5.065678e-02
1.500000e-02 9.182239e-02 7.959676e-02
5.000000e-02 3.696508e-01 3.220606e-01

```

```

5
14
1.000000e-03 3.808086e-05 2.680139e-05
2.000000e-03 3.165315e-04 2.257910e-04
5.000000e-03 5.035972e-03 3.770583e-03
6.500000e-03 1.064455e-02 8.347346e-03
7.000000e-03 1.359007e-02 1.060487e-02
7.500000e-03 1.654580e-02 1.294715e-02
8.000000e-03 1.929566e-02 1.577339e-02
8.500000e-03 2.354591e-02 1.927205e-02
9.000000e-03 2.829314e-02 2.309704e-02
9.500000e-03 3.226927e-02 2.678389e-02
1.000000e-02 3.675606e-02 3.050584e-02
1.200000e-02 6.141257e-02 5.081799e-02
1.500000e-02 1.106172e-01 9.819651e-02
5.000000e-02 4.610004e-01 4.200611e-01
7
14
1.000000e-03 2.294325e-06 1.335142e-06
2.000000e-03 4.335013e-05 2.570402e-05
5.000000e-03 2.094460e-03 1.429906e-03
6.500000e-03 6.262974e-03 4.548905e-03
7.000000e-03 8.185048e-03 5.987798e-03
7.500000e-03 1.096092e-02 8.152393e-03
8.000000e-03 1.433186e-02 1.116570e-02
8.500000e-03 1.838769e-02 1.426086e-02
9.000000e-03 2.288849e-02 1.804973e-02
9.500000e-03 2.723837e-02 2.213544e-02
1.000000e-02 3.430413e-02 2.778531e-02
1.200000e-02 7.393956e-02 6.362081e-02
1.500000e-02 1.336675e-01 1.214404e-01
5.000000e-02 4.826679e-01 4.646626e-01
9
13
2.000000e-03 5.701198e-06 2.960670e-06
5.000000e-03 9.323964e-04 5.990253e-04
6.500000e-03 3.723894e-03 2.571852e-03
7.000000e-03 5.436050e-03 3.922925e-03
7.500000e-03 7.731204e-03 5.614499e-03
8.000000e-03 1.103955e-02 8.423489e-03
8.500000e-03 1.475789e-02 1.119869e-02
9.000000e-03 1.907401e-02 1.465237e-02
9.500000e-03 2.606438e-02 2.132012e-02
1.000000e-02 3.509833e-02 2.894034e-02
1.200000e-02 8.033013e-02 7.130980e-02
1.500000e-02 1.657271e-01 1.449695e-01
5.000000e-02 4.954739e-01 4.866886e-01
11
13
2.000000e-03 7.559446e-07 3.382315e-07
5.000000e-03 4.250431e-04 2.457825e-04
6.500000e-03 2.319008e-03 1.527896e-03
7.000000e-03 3.631596e-03 2.525026e-03
7.500000e-03 5.625133e-03 3.980921e-03
8.000000e-03 8.302013e-03 6.213838e-03
8.500000e-03 1.218413e-02 9.219708e-03
9.000000e-03 1.886069e-02 1.443793e-02
9.500000e-03 2.490968e-02 2.022095e-02
1.000000e-02 3.029043e-02 2.499077e-02
1.200000e-02 9.084678e-02 7.889485e-02
1.500000e-02 1.921186e-01 1.719365e-01
5.000000e-02 4.971981e-01 4.999981e-01
13
12
5.000000e-03 1.875611e-04 1.023601e-04
6.500000e-03 1.487202e-03 9.451868e-04
7.000000e-03 2.534618e-03 1.704212e-03
7.500000e-03 4.172993e-03 2.876705e-03
8.000000e-03 6.748340e-03 4.856594e-03
8.500000e-03 1.026050e-02 7.700932e-03

```

9.000000e-03 1.597595e-02 1.217906e-02
 9.500000e-03 2.240651e-02 1.773543e-02
 1.000000e-02 3.281617e-02 2.687198e-02
 1.200000e-02 1.036773e-01 8.689857e-02
 1.500000e-02 2.194815e-01 1.988821e-01
 5.000000e-02 4.999981e-01 4.999981e-01
 15
 12
 5.000000e-03 8.458677e-05 4.400683e-05
 6.500000e-03 9.483575e-04 5.813465e-04
 7.000000e-03 1.807830e-03 1.156577e-03
 7.500000e-03 3.220028e-03 2.172807e-03
 8.000000e-03 5.559875e-03 3.862835e-03
 8.500000e-03 8.742246e-03 6.528671e-03
 9.000000e-03 1.494599e-02 1.154505e-02
 9.500000e-03 2.272859e-02 1.804911e-02
 1.000000e-02 3.381841e-02 2.723163e-02
 1.200000e-02 1.128764e-01 9.816980e-02
 1.500000e-02 2.455072e-01 2.260046e-01
 5.000000e-02 4.985409e-01 4.999981e-01
 17
 12
 5.000000e-03 3.907789e-05 1.862184e-05
 6.500000e-03 6.127439e-04 3.577438e-04
 7.000000e-03 1.287325e-03 8.089195e-04
 7.500000e-03 2.439529e-03 1.638397e-03
 8.000000e-03 4.523496e-03 3.145585e-03
 8.500000e-03 7.664518e-03 5.666593e-03
 9.000000e-03 1.432955e-02 1.090556e-02
 9.500000e-03 2.227061e-02 1.779357e-02
 1.000000e-02 3.680715e-02 3.044608e-02
 1.200000e-02 1.268430e-01 1.111875e-01
 1.500000e-02 2.775030e-01 2.547770e-01
 5.000000e-02 4.905796e-01 4.996738e-01
 19
 10
 6.500000e-03 4.000280e-04 2.265075e-04
 7.000000e-03 9.146123e-04 5.520112e-04
 7.500000e-03 1.953781e-03 1.254145e-03
 8.000000e-03 3.708987e-03 2.579145e-03
 8.500000e-03 6.906109e-03 4.763698e-03
 9.000000e-03 1.230079e-02 9.332162e-03
 9.500000e-03 2.253916e-02 1.807942e-02
 1.000000e-02 3.592503e-02 2.861376e-02

1.200000e-02 1.407514e-01 1.233468e-01
 1.500000e-02 3.040190e-01 2.724428e-01
 21
 10
 6.500000e-03 2.641856e-04 1.436088e-04
 7.000000e-03 6.622730e-04 3.862715e-04
 7.500000e-03 1.521229e-03 9.543867e-04
 8.000000e-03 3.183285e-03 2.101901e-03
 8.500000e-03 6.229161e-03 4.419605e-03
 9.000000e-03 1.195018e-02 9.221913e-03
 9.500000e-03 2.425194e-02 1.899817e-02
 1.000000e-02 3.669981e-02 3.066557e-02
 1.200000e-02 1.503129e-01 1.321197e-01
 1.500000e-02 3.330526e-01 2.998056e-01
 25
 5
 8.000000e-03 2.291625e-03 1.534796e-03
 8.500000e-03 5.385383e-03 3.781245e-03
 9.000000e-03 1.119275e-02 8.279782e-03
 9.500000e-03 2.501067e-02 2.004721e-02
 1.000000e-02 0.000000e+00 1.111112e-01
 35
 5
 8.000000e-03 1.168454e-03 7.215547e-04
 8.500000e-03 3.946915e-03 2.750413e-03
 9.000000e-03 1.105951e-02 8.494526e-03
 9.500000e-03 2.666542e-02 2.130889e-02
 1.000000e-02 4.625748e-02 1.492872e-02
 45
 5
 8.000000e-03 6.210500e-04 3.678873e-04
 8.500000e-03 2.746342e-03 1.915113e-03
 9.000000e-03 1.060803e-02 8.018221e-03
 9.500000e-03 3.327366e-02 2.818675e-02
 1.000000e-02 0.000000e+00 7.692313e-02
 55
 5
 8.000000e-03 3.514479e-04 1.940539e-04
 8.500000e-03 2.146034e-03 1.463901e-03
 9.000000e-03 1.126806e-02 8.856805e-03
 9.500000e-03 4.394984e-02 3.685991e-02
 1.000000e-02 1.363635e-01 1.515150e-02

-
- [1] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, arXiv:1110.5133 (2011).
 [2] S. B. Bravyi and A. Y. Kitaev, quant-ph/9811052 (1998).
 [3] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, J. Math. Phys. **43**, 4452 (2002), quant-ph/0110143.
 [4] R. Raussendorf and J. Harrington, Phys. Rev. Lett. **98**, 190504 (2007), quant-ph/0610082.
 [5] R. Raussendorf, J. Harrington, and K. Goyal, New J.

- Phys. **9**, 199 (2007), quant-ph/0703143.
 [6] A. G. Fowler, A. M. Stephens, and P. Groszkowski, Phys. Rev. A **80**, 052312 (2009), arXiv:0803.0272.
 [7] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg, arXiv:1009.3686 (2010).
 [8] J. Edmonds, Canad. J. Math. **17**, 449 (1965).
 [9] J. Edmonds, J. Res. Nat. Bur. Standards **69B**, 125 (1965).